Introduction to SQL

Class 14

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, REPEAT, LOOP), ,Stored functions ,Stored Procedures
  - Views
  - Indexes, Cursors, Triggers, Events
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases
  - ACID, CAP
  - How SQL databases internally works

TRANSACTIONS

- A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit
- In other words, a transaction will never be complete unless each individual operation within the group is successful
- If any operation within the transaction fails, the entire transaction will fail.

Real life examples of transactions

- Banking operations: Transactions can be used to ensure that banking operations, such as transferring funds from one account to another, are executed atomically and in a consistent manner.
- Online shopping: Transactions can be used to ensure that a customer's order is properly recorded in the database, including updating the inventory and creating an order record. This way, even if there is a failure in the middle of the process, the transaction can be rolled back to avoid partial updates or inconsistencies in the database.
- Stock market trading: Transactions can be used to ensure that a stock trade is executed atomically and in a consistent manner. For example, if a trade involves buying and selling stocks, the transaction ensures that either both operations succeed or both fail, avoiding partial updates or inconsistencies in the database.
- Flight booking: Transactions can be used to ensure that a flight booking is properly recorded in the database, including updating the available seats and creating a booking record. This way, even if there is a failure in the middle of the process, the transaction can be rolled back to avoid partial updates or inconsistencies in the database.
- Health care management: Transactions can be used to ensure that patient information is recorded atomically and in a consistent manner, avoiding partial updates or inconsistencies in the database.

Properties of transactions

- Transactions have the following four standard properties, usually referred to by the acronym **ACID**
- **Atomicity** – This ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.
- **Consistency** – This ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation** – This enables transactions to operate independently on and transparent to each other.
- **Durability** – This ensures that the result or effect of a committed transaction persists in case of a system failure

Atomicity

- This property ensures that all statements or operations within the transaction unit must be executed successfully. Otherwise, if any operation is failed, the whole transaction will be aborted, and it goes rolled back into their previous state. It includes features:
  - COMMIT statement.
  - ROLLBACK statement.
  - Auto-commit setting.
  - Operational data from the INFORMATION_SCHEMA tables.

Consistency

- This property ensures that the database changes state only when a transaction will be committed successfully. It is also responsible for protecting data from crashes. It includes features:
  - InnoDB doublewrite buffer.
  - InnoDB crash recovery.

Isolation

- This property guarantees that each operation in the transaction unit operated independently. It also ensures that statements are transparent to each other. It includes features:
  - SET ISOLATION LEVEL statement.
  - Auto-commit setting.
  - The low-level details of InnoDB locking.

Durability

- This property guarantees that the result of committed transactions persists permanently even if the system crashes or failed. It includes features:
  - Write buffer in a storage device.
  - Battery-backed cache in a storage device.
  - Configuration option innodb_file_per_table.
  - Configuration option innodb_flush_log_at_trx_commit.
  - Configuration option sync_binlog.

COMMIT,ROLLBACK,SAVEPOINT

- In MySQL, the transactions begin with the statement **BEGIN WORK** and end with either a **COMMIT** or a **ROLLBACK** statement
- The SQL commands between the beginning and ending statements form the bulk of the transaction
- **COMMIT** - When a successful transaction is completed, the COMMIT command should be issued so that the changes to all involved tables will take effect.
- **ROLLBACK** - If a failure occurs, a ROLLBACK command should be issued to return every table referenced in the transaction to its previous state.

AUTOCOMMIT

- You can control the behavior of a transaction by setting session variable called AUTOCOMMIT
- If AUTOCOMMIT is set to 1 (the default), then each SQL statement (within a transaction or not) is considered a complete transaction and committed by default when it finishes
- When AUTOCOMMIT is set to 0, by issuing the **SET AUTOCOMMIT = 0** command, the subsequent series of statements acts like a transaction and no activities are committed until an explicit COMMIT statement is issued.

A Generic Example on Transaction

- Begin transaction by issuing the SQL command BEGIN WORK.
- Issue one or more SQL commands like SELECT, INSERT, UPDATE or DELETE.
- Check if there is no error and everything is according to your requirement.
- If there is any error, then issue a ROLLBACK command, otherwise issue a COMMIT command.

COMMIT Example

```
-- 1. Start a new transaction
START TRANSACTION;
-- 2. Get the highest income
SELECT @income:= MAX(income) FROM employees;
-- 3. Insert a new record into the employee table
INSERT INTO employees(emp_id, emp_name, emp_age, city, income)
VALUES (111, 'Alexander', 45, 'California', 70000);
-- 4. Insert a new record into the order table
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');
-- 5. Commit changes
COMMIT;
```

ROLLBACK Example

```
-- 1. Start a new transaction
START TRANSACTION;
-- 2. Delete data from the order table
DELETE FROM Orders;
-- 3. Rollback changes
ROLLBACK;
-- 4. Verify the records in the first session
SELECT * FROM Orders;
```

Statements that cannot be a rollback in using MySQL Transaction.

- MySQL Transaction cannot be able to roll back all statements. For example, these statements include DDL (Data Definition Language) commands such as CREATE, ALTER, or DROP database as well as CREATE, UPDATE, or DROP tables or stored routines. We have to make sure that when we design our transaction, these statements do not include.

SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT

- The SAVEPOINT statement creates a special mark with the name of the identifier inside a transaction.
- It allows all statements that are executed after savepoint to be rolled back. So that the transaction restores to the previous state it was in at the point of the savepoint
- The ROLLBACK TO SAVEPOINT statement allows us to rolls back all transactions to the given savepoint was established without aborting the transaction.
- The RELEASE SAVEPOINT statement destroys the named savepoint from the current transaction without undoing the effects of queries executed after the savepoint was established.

SAVEPOINT savepoint_name

ROLLBACK TO [SAVEPOINT] savepoint_name

RELEASE SAVEPOINT savepoint_name

ROLLBACK TO SAVEPOINT example

```
START TRANSACTION;
SELECT * FROM Orders;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');
SAVEPOINT my_savepoint;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (7, 'Ink', 5894, '2020-03-10');
ROLLBACK TO SAVEPOINT my_savepoint;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (8, 'Speaker', 6065, '2020-02-18');
COMMIT;
```

RELEASE SAVEPOINT

```
START TRANSACTION;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (7, 'Ink', 5894, '2020-03-10');
SAVEPOINT my_savepoint;
UPDATE Orders SET prod_name='Scanner' WHERE order_id=8;
RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

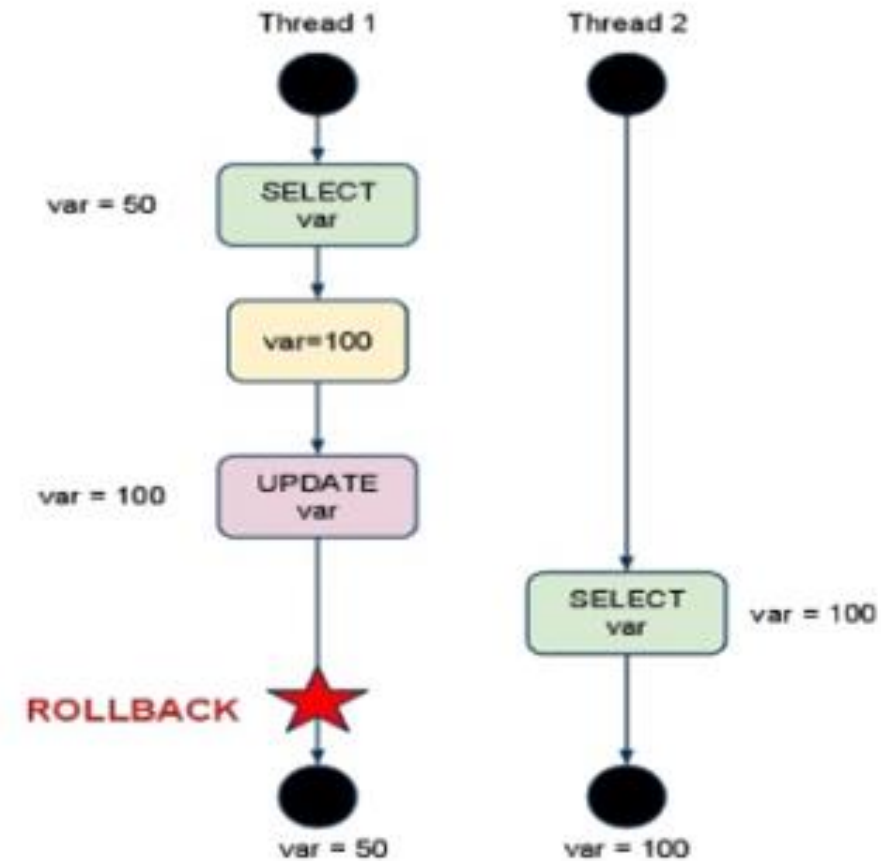
Isolation Levels for a transaction In MySQL

- There are 4 levels of isolation supported by mysql
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
- The Isolation level’s can be set globally or session based on our requirements.

Read Uncommitted

- In READ-UNCOMMITTED isolation level, there isn’t much isolation present between the transactions at all, ie ., No locks
- A transaction can see changes to data made by other transactions that are not committed yet
- A transaction can see changes to data made by other transactions that are not committed yet
- With this isolation level, there is always for getting a “Dirty-Read”

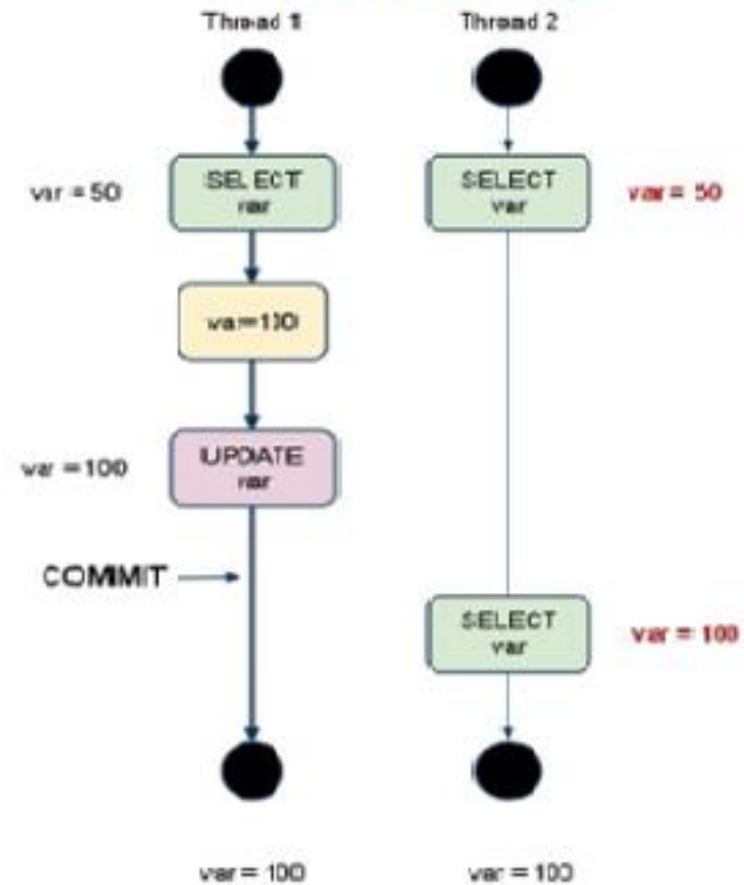
DIRTY READS


Thread 1
Thread 2
var = 50
SELECT var
var=100
var = 100
UPDATE var
SELECT var
var = 100
ROLLBACK
var = 50
var = 100

READ COMMITTED

- IN READ-COMMITTED isolation level, the phenomenon of dirty read is avoided, because any uncommitted changes are not visible to any other transaction until the change is committed
- This is the default isolation level with most of popular RDBMS software, but not with MySQL.
- Within this isolation level, each SELECT uses its own snapshot of the committed data that was committed before the execution of the SELECT
- Now because each SELECT has its own snapshot, here is the trade-off now, so the same SELECT, when running multiple times during the same transaction, could return different result sets. This phenomenon is called non-repeatable read.

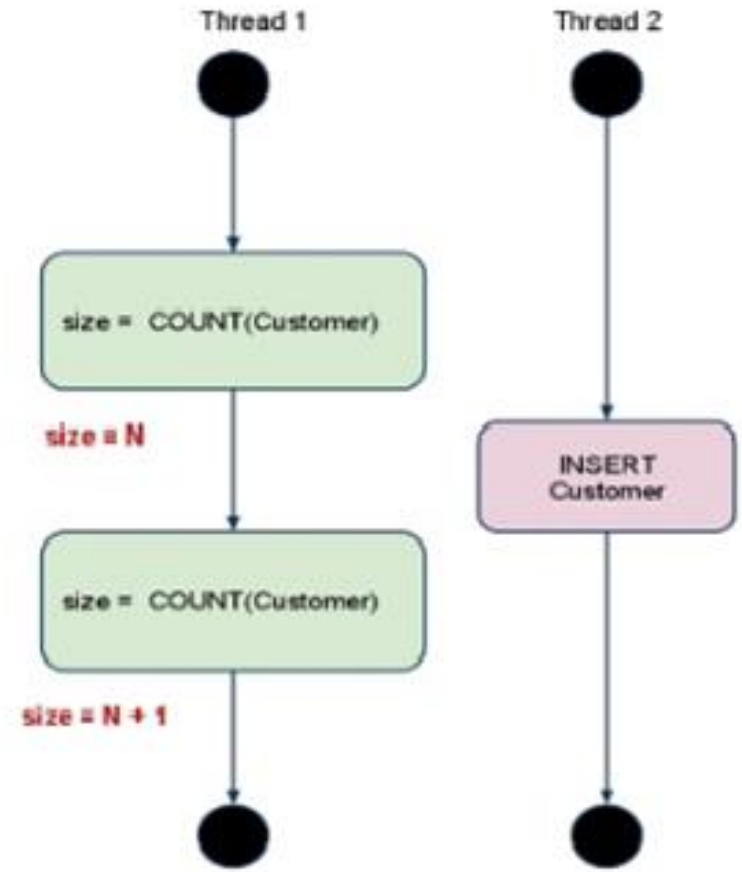
NON REPEATABLE READS



REPEATABLE READ

- In REPEATABLE-READ isolation level, the phenomenon of non-repeatable read is avoided.
- It is the default isolation in MySQL.
- This isolation level returns the same result set throughout the transaction execution for the same SELECT run any number of times during the progression of a transaction.
- A snapshot of the SELECT is taken the first time the SELECT is run during the transaction and the same snapshot is used throughout the transaction when the same SELECT is executed.
- A transaction running in this isolation level does not take into account any changes to data made by other transactions, regardless of whether the changes have been committed or not.
- This ensures that reads are always consistent(repeatable)
- Maintaining a snapshot can cause extra overhead and impact some performance
- Although this isolation level solves the problem of non-repeatable read, another possible problem that occurs is phantom reads.

PHANTOM READS



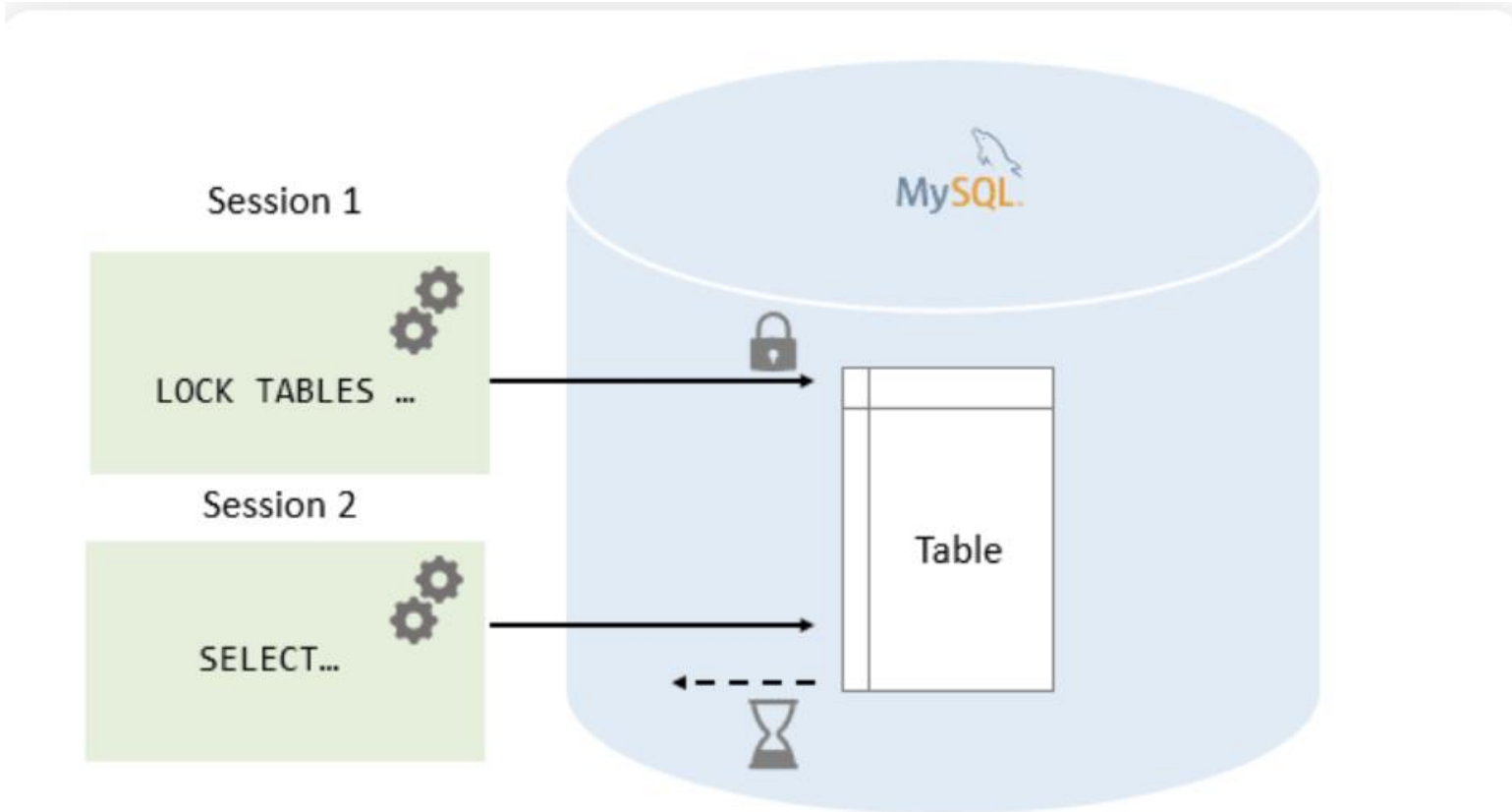
SERIALIZABLE

- SERIALIZABLE completely isolates the effect of one transaction from others
- It is similar to REPEATABLE READ with the additional restriction that row selected by one transaction cannot be changed by another until the first transaction finishes
- The phenomenon of phantom reads is avoided
- This isolation level is the strongest possible isolation level

Catching errors and then rolling back

```
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
      BEGIN
         ROLLBACK;
         RESIGNAL;
      END;

   START TRANSACTION;
      -- .. Query 1 ..
      -- .. Query 2 ..
      -- .. Query 3 ..
   COMMIT;
END;
```


Session 1
LOCK TABLES ...
Session 2
SELECT...
MySQL
Table

What are MySQL Locks?

- A MySQL Locks is nothing but a flag that can be assigned to a table to alter its properties. MySQL allows a table lock that can be assigned by a client-server to prevent other sessions from being able to access the same table during a specific time frame.

LOCK TABLES

table_name [[AS] alias] lock_type

[, tbl_name [[AS] alias] lock_type] ...

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

lock a table using an alias, you must refer to it in your statements using that alias

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES

mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

What are 2 Types of MySQL Locks?

- MySQL Locks: Read Locks
- MySQL Locks: Write Locks

Features of the READ Lock

- MySQL allows multiple sessions to bi in READ lock for a table concurrently. And it also allows for other sessions to read the table without acquiring the lock.
- If the session holds the READ lock on a table, they cannot perform a write operation on it. It is because the READ lock can only read data from the table. All other sessions that do not acquire a READ lock are not able to write data into the table without releasing the READ lock. The write operations go into the waiting states until we have not released the READ lock.
- When the session is terminated normally or abnormally, MySQL implicitly releases all types of locks onto the table. This feature is also relevant for the WRITE lock

- WRITE locks have higher priority when compared to READ locks to ensure that updates are processed asap.
- This means that if a session has obtained a READ lock and simultaneously another session requests a WRITE lock, the session with READ lock requests has to wait until the session that requested the WRITE the lock has obtained the lock and released it.

Features of WRITE locks

- It is the session that holds the lock of a table and can read and write data both from the table.
- It is the only session that accesses the table by holding a lock. And all other sessions cannot access the data of the table until the WRITE lock is released.

How to unlock the table in MySQL?

- If a session issues a LOCK TABLES statement to acquire a lock while already holding MySQL locks, its existing locks are released implicitly before the new locks get granted.
- A session can release its locks explicitly with the help of UNLOCK TABLES.
- If a session begins a transaction, an implicit UNLOCK TABLES is performed, which causes existing MySQL locks to be released.
- If you lock a table explicitly with LOCK TABLES, any tables used in triggers also get locked implicitly:
- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.

Data modelling and Database design

What is a data model?

- A data model is a simplified representation of a real-world system or phenomenon that is used to describe and understand the data
- It is a way of organizing and structuring data to effectively and efficiently store, process, and retrieve information.
- Data models can take many forms, such as a relational database model, a network model, an object-oriented model, and more.

Objectives

- Identify and describe important entities and relationships to model data
- Develop data models to represent, organize, and store data
- Design and use relational databases to organize, store, and manipulate data

Data Modeling

- Goal – make sure all data objects required by a database are completely and accurately represented
- Data model design – the blueprint for creating a physical implementation of a database

Data Modeling Terms

- **Entity** – a class of real world objects having common attributes (e.g., sites, variables, methods).
- **Attribute** – A characteristic or property of an entity (site name, latitude, longitude)
- **Relationship** – an association between two or more entities
- **Cardinality** – the number of entities on either end of a relationship (one-to-one, one-to-many, many-to-many, etc.)

Data Modeling Exercise

- Consider:
  - What is the “entity”?
  - What are the “attributes” the entity?

Data Modeling Exercise

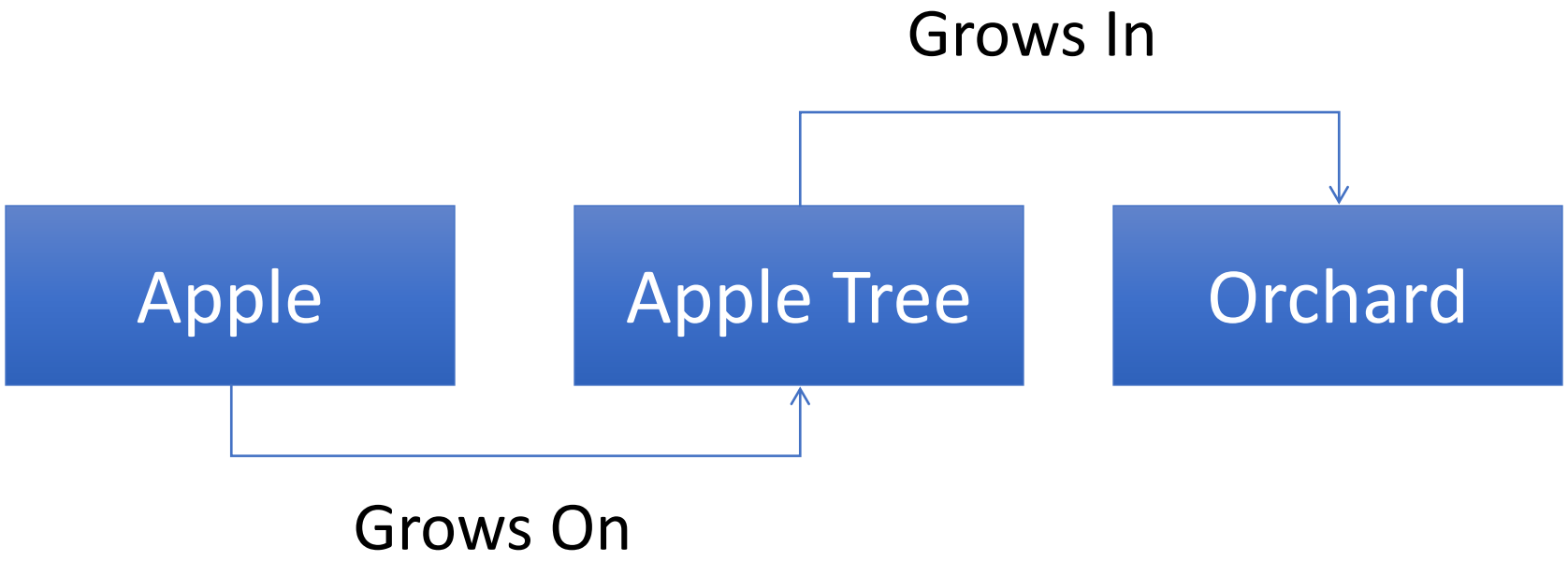
- What is the entity?
- What are the attributes?

Data Modeling Exercise

- What is the entity?
- What are the attributes?

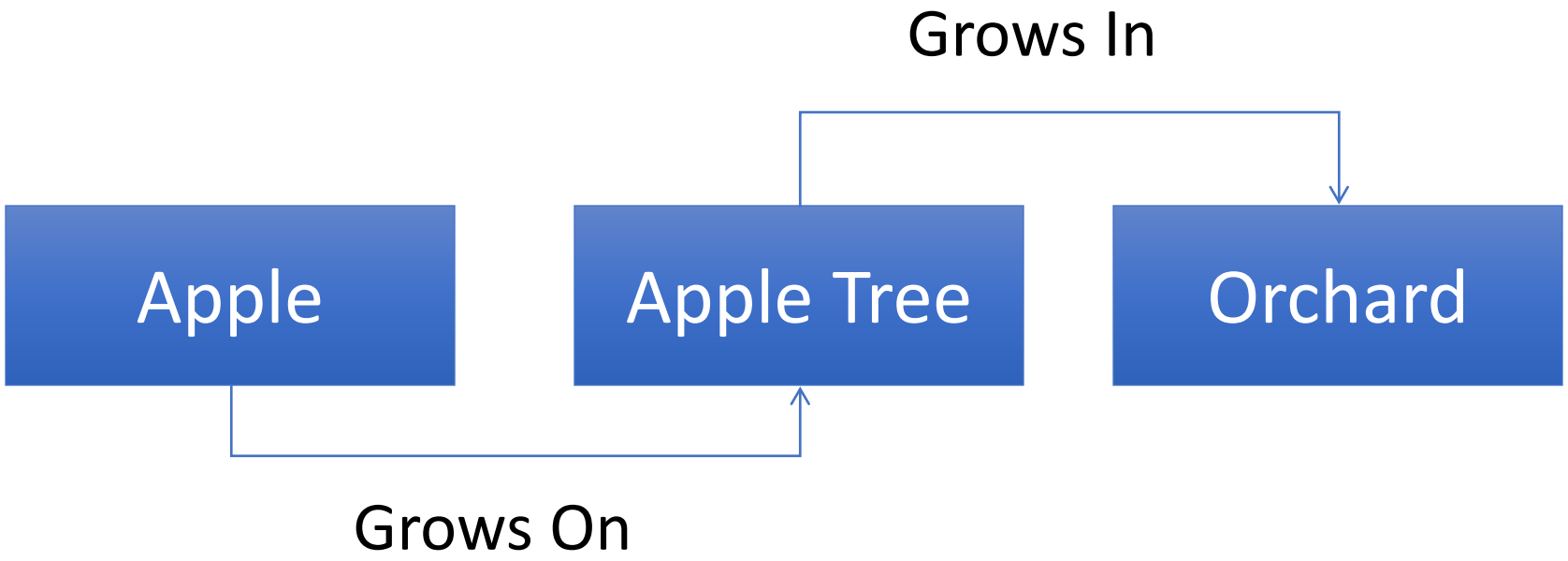
Data Modeling Exercise

- What are the relationships?



Data Modeling Exercise

- What are the relationships?



What about the business rules?

Data Model Requirements

- What is the information/data domain that you are modeling?
- What are the 20 queries that you want to do?
  - e.g., "Give me simultaneous observations of turbidity and TSS collected during the spring snowmelt period so I can develop a regression in R."
- What software do you want (have) to use?
- How do you want to share the data?

Data Model Design

- Our focus – relational data model design
- Three stages:
  - Conceptual data model
  - Logical data model
  - Physical data model

Conceptual Data Model (AKA – The Information Model)

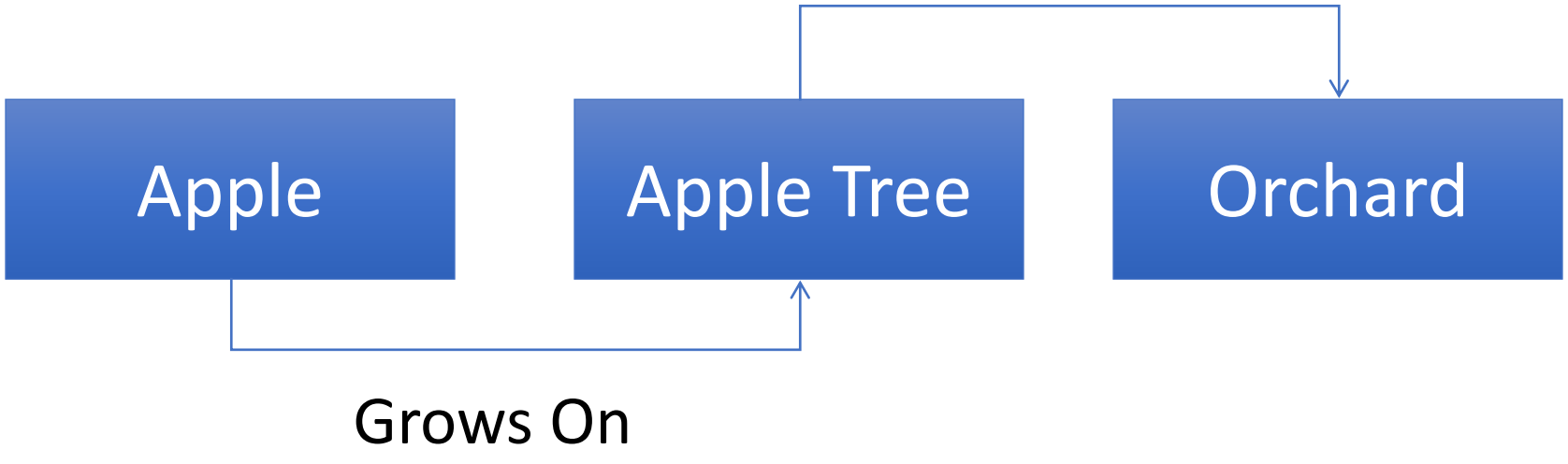
- High-level description of the data domain
- Does not constrain how that description is mapped to an actual implementation in software
- There may be many mappings
  - Relational database
  - Object model
  - XML schema, etc.

Goals of Conceptual Data Modeling

- The goals of conceptual data modeling are:
  - To capture the essential and intrinsic characteristics of data, independent of the physical storage or technology used to manage the data.
  - To create a common understanding and representation of the data among stakeholders, such as business analysts, data analysts, and IT personnel.
  - To identify and define the relationships and dependencies between data entities.
  - To support the design of effective and efficient data storage, retrieval, and processing systems.
  - To support the evolution and growth of the data and information systems over time.
- Overall, conceptual data modeling is a critical step in the data management process that helps organizations understand and manage their data more effectively.

Apple/Tree/Orchard Conceptual Model

Grows In



Logical Data Model

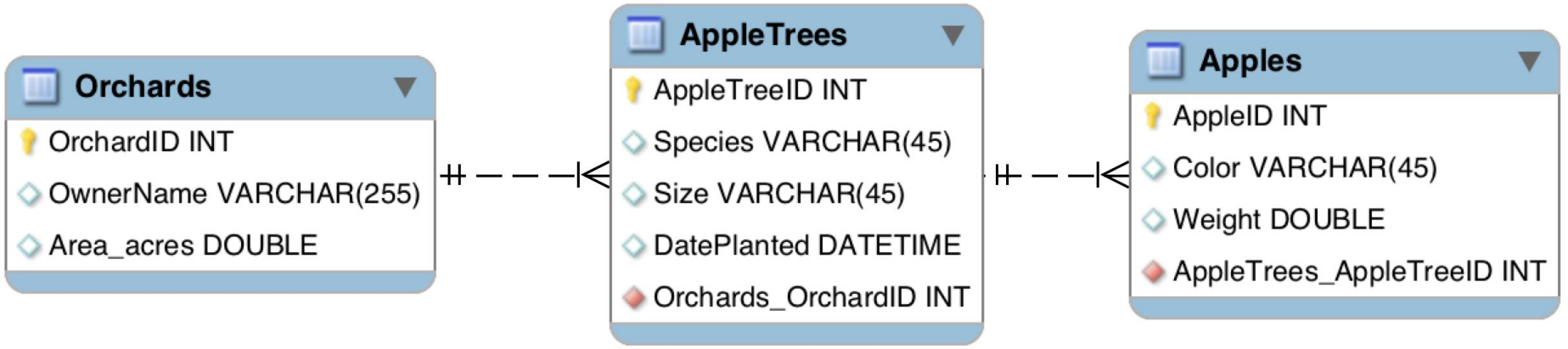
- Technology independent
- Contains more detail than the Conceptual Data Model
- Considered by many to be just an expanded conceptual data model
- Defines
  - Entities AND their attributes
  - Relationships AND cardinality
  - Constraints
- Generally completed as a documented Entity Relationship (ER) Model or diagram

Goals of Logical Data Modeling

- The goals of logical data modeling are:
  - To refine the conceptual data model and define a more detailed representation of the data, including specific data elements, data types, and relationships between entities.
  - To ensure that the data model accurately reflects the business requirements and meets the needs of the stakeholders.
  - To provide a consistent and unambiguous representation of the data that can be used as a blueprint for physical database design and implementation.
  - To provide a clear definition of the business rules and constraints that apply to the data.
  - To facilitate the integration of data from multiple sources into a unified data model.
- Overall, logical data modeling is a critical step in the data management process that helps organizations design and implement effective and efficient data management systems.

Entity Relationship Diagram

- Documentation of the structure of the data
- Used to communicate the design
- Serve as the basis for data model implementation



Advantages of having Entity Relationship Diagram

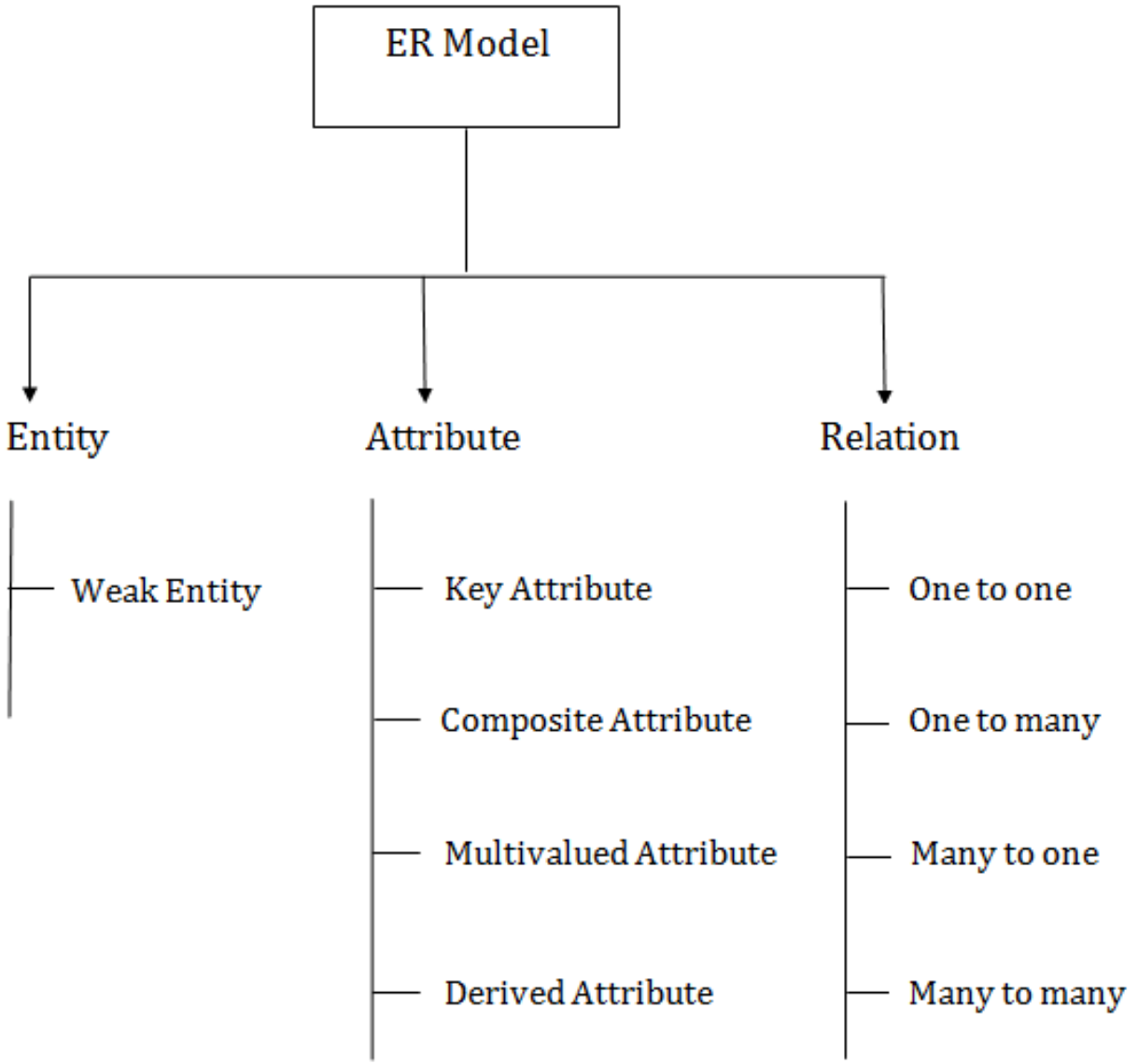
- The Entity Relationship (ER) diagram has several advantages, including:
  - Improved Communication: ER diagrams provide a visual representation of the data model that can be easily understood by stakeholders, including business analysts, data analysts, and IT personnel. This improved communication helps to ensure that the data model accurately reflects the business requirements.
  - Better Data Modeling: ER diagrams provide a systematic and structured approach to data modeling that helps to identify entities, relationships, and constraints. This leads to a more accurate and complete representation of the data.
  - Enhanced Data Integrity: ER diagrams define the relationships between entities, which helps to ensure that the data is stored and processed in a consistent and correct manner. This leads to enhanced data integrity and reduces the risk of data inconsistencies and errors.
  - Better Database Design: ER diagrams provide a basis for physical database design and implementation. They help to ensure that the database design is optimized for performance and scalability, and that it meets the needs of the stakeholders.
  - Improved Data Management: ER diagrams provide a clear and concise representation of the data that can be used as a reference for data management activities, such as data warehousing, data integration, and data quality management.
- Overall, ER diagrams provide a powerful tool for data modeling and management that can help organizations to better understand and manage their data.

Entity Relationship Diagram
(Relation Database Context)

- Entities effectively become tables
- Attributes describe entities and become fields (columns) in tables
- Relationships link tables on a common attribute or “key” and become formal constraints (part of the business rules)

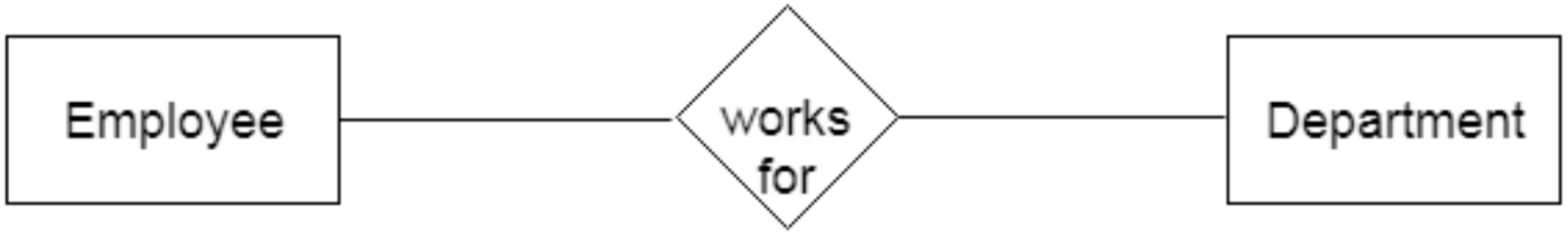
Main Elements of an Entity-Relationship Diagram

- Entities: Entities are objects or concepts that exist in the real-world and have a unique identity. They are represented as boxes or rectangles in an ER diagram.
- Attributes: Attributes are characteristics or properties of entities. They are represented as ellipses within the entity rectangle in an ER diagram.
- Relationships: Relationships describe the association between two or more entities. They are represented as a diamond shape connecting two or more entities in an ER diagram.
- Cardinality: Cardinality defines the number of instances of one entity that are associated with one instance of another entity. It is represented as an arrowhead on the relationship line in an ER diagram.
- Optionality: Optionality defines whether the relationship between two entities is mandatory or optional. It is represented by a circle or an open or filled diamond at the end of the relationship line in an ER diagram.
- Key Constraints: Key constraints define the unique identifier for each entity. They are represented by underlining the attribute name in an ER diagram.


ER Model
Entity
Attribute
Relation
Weak Entity
Key Attribute
Composite Attribute
Multivalued Attribute
Derived Attribute
One to one
One to many
Many to one
Many to many

ER Diagram Entity Notation

ENTITY



Regular or Strong & Weak entities (Independent and dependent entities)

- Strong or Regular Entities: Strong entities have a unique identifier called a primary key and can exist independently in the database. They do not rely on other entities to identify or describe them.
- Weak Entities: Weak entities do not have a primary key and cannot exist independently in the database. They are dependent on another entity, called the owner entity, to identify or describe them. The combination of the primary key of the owner entity and the weak entity's partial key forms a unique identifier for the weak entity.

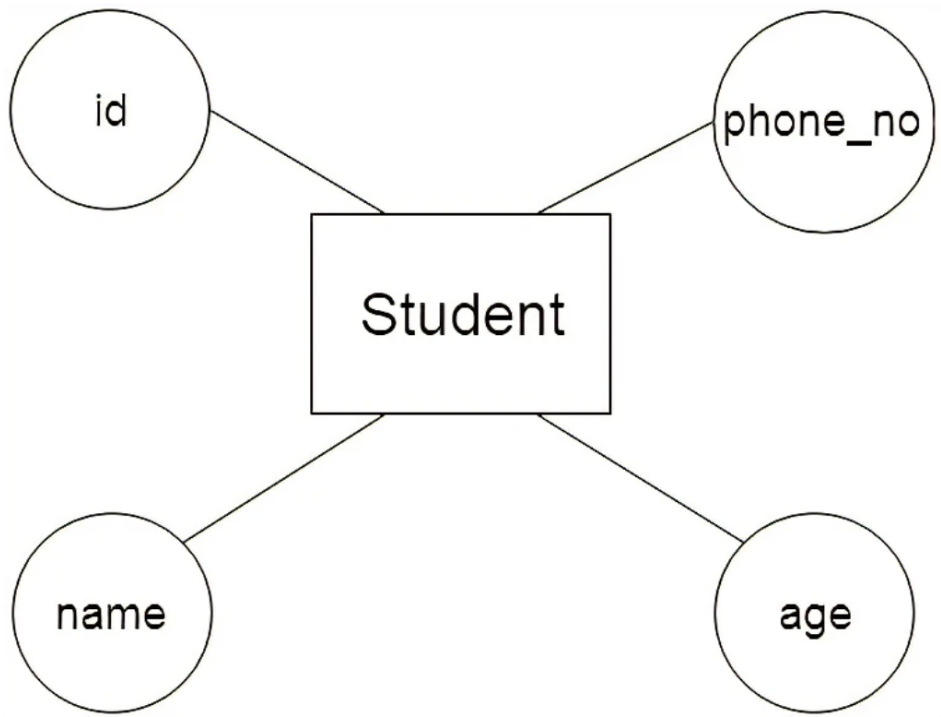
a. Weak Entity

An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.

2. Attribute

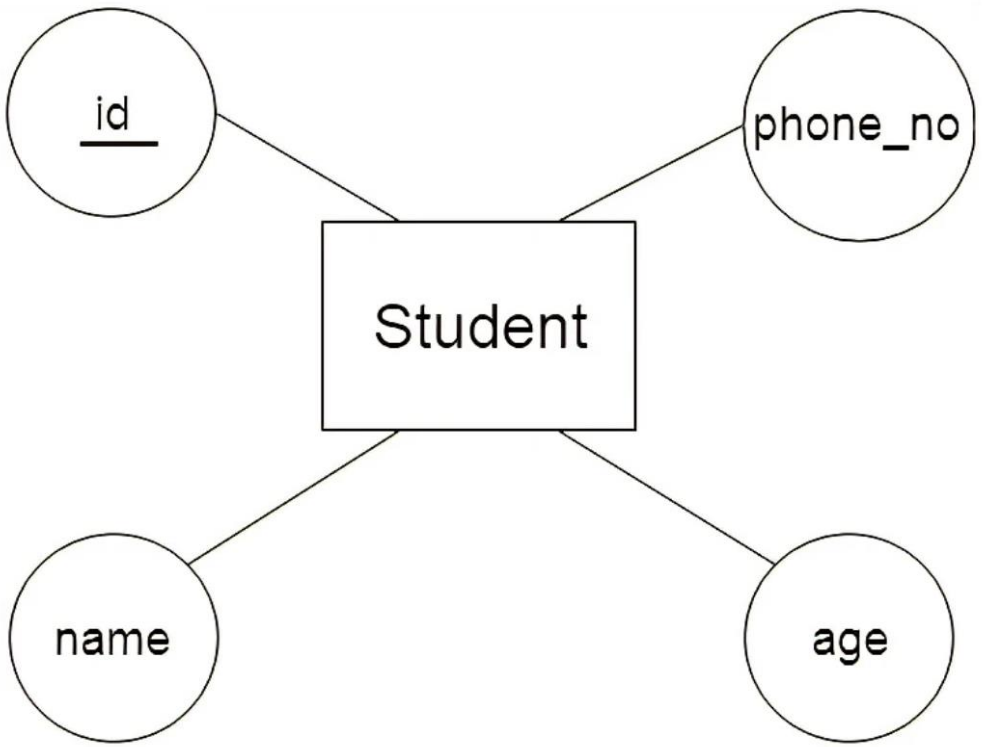
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



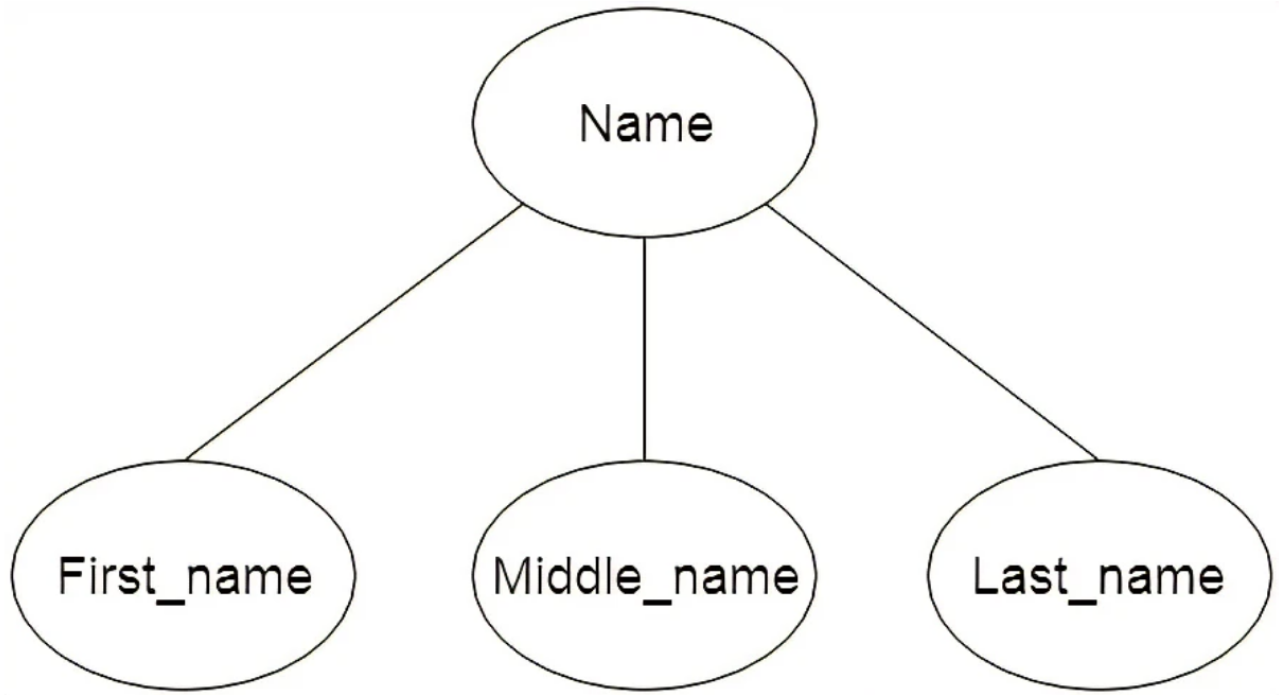
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

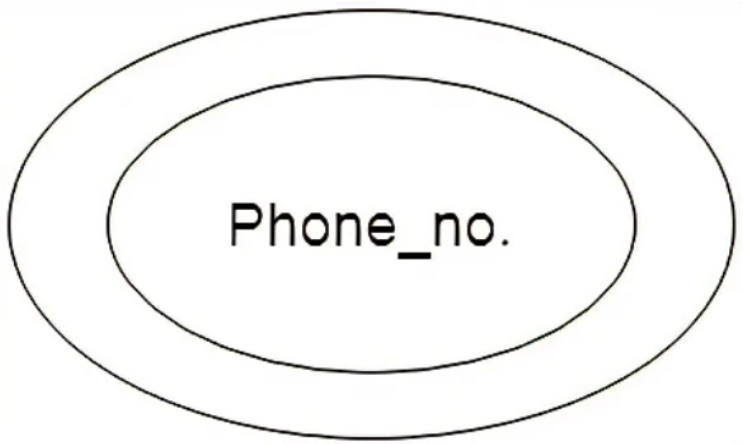
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

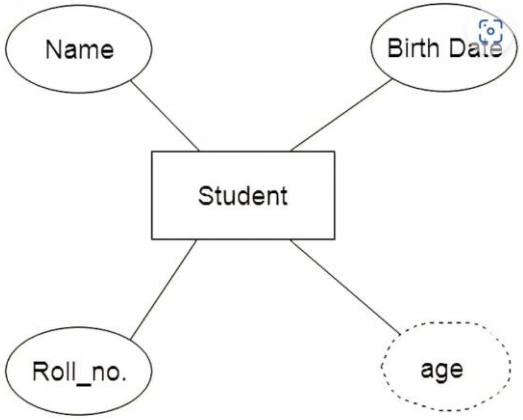
For example, a student can have more than one phone number.



d. Derived Attribute

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

a. One-to-One Relationship

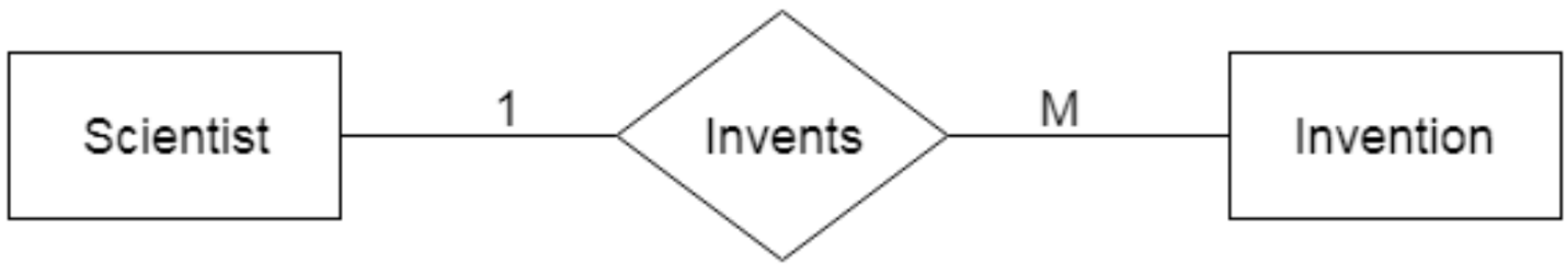
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

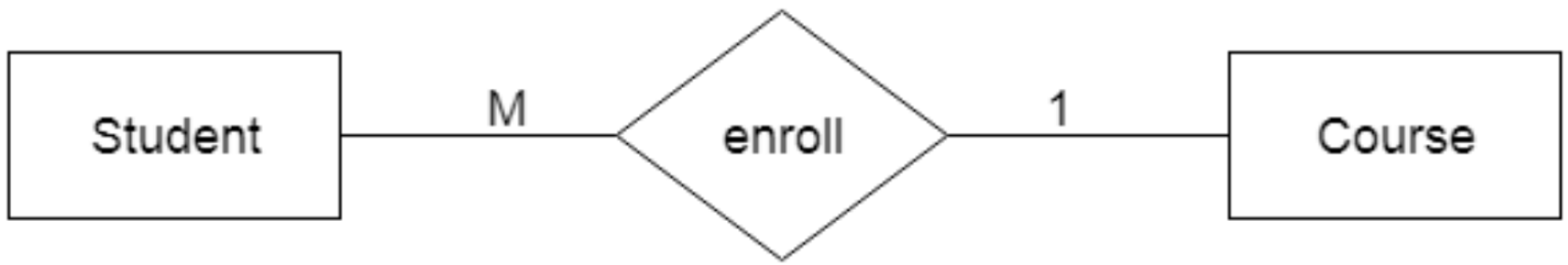
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

Summary

- Data model design is a 3 step process – conceptual, logical, physical (next time)
- Conceptual and logical data models can be expressed using Entity Relationship (ER) diagrams
- ER diagrams capture the entities, attributes, and relationships to model your information domain
- ER diagrams are a powerful way to document the design of your data model

Steps in Data Model Design

1. Identify entities
2. Identify relationships among entities
3. Determine the cardinality and participation of relationships
4. Designate keys / identifiers for entities
5. List attributes of entities
6. Identify constraints and business rules

ER Modeling Case Studies

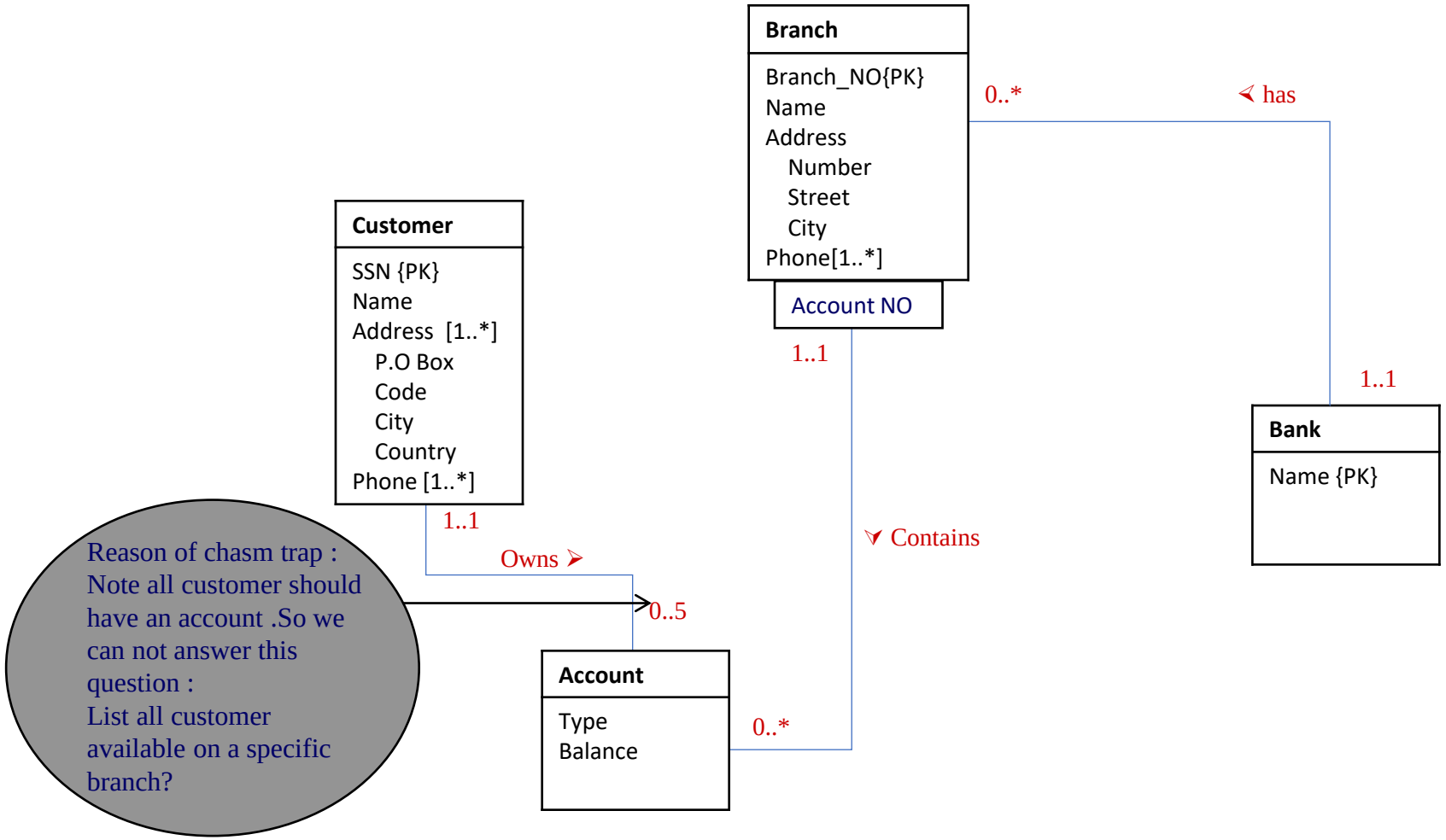
BanksDatabase :ER Case Study

- Consider the following set of requirements for a Bank database that is used to keep track of Customer.
  a) Each bank has a unique name.
  b) Each branch has a number, name, address (number, street, city), and set of phones.
  c) Customer includes their name, set of address (P.O. Box, city, zip code, country), set of phones, and social security number.
  d) Accounts have numbers, types (e.g. saving, checking) and balance. Other branches might use the same designation for accounts. So to name an account uniquely, we need to give both the branch number to which this account belongs to and the account number.
  e) Not all bank customers must own accounts and a customer may have at most 5 accounts in the bank.
  f) An account must have only one customer.
  g) A customer may have many accounts in different branches.

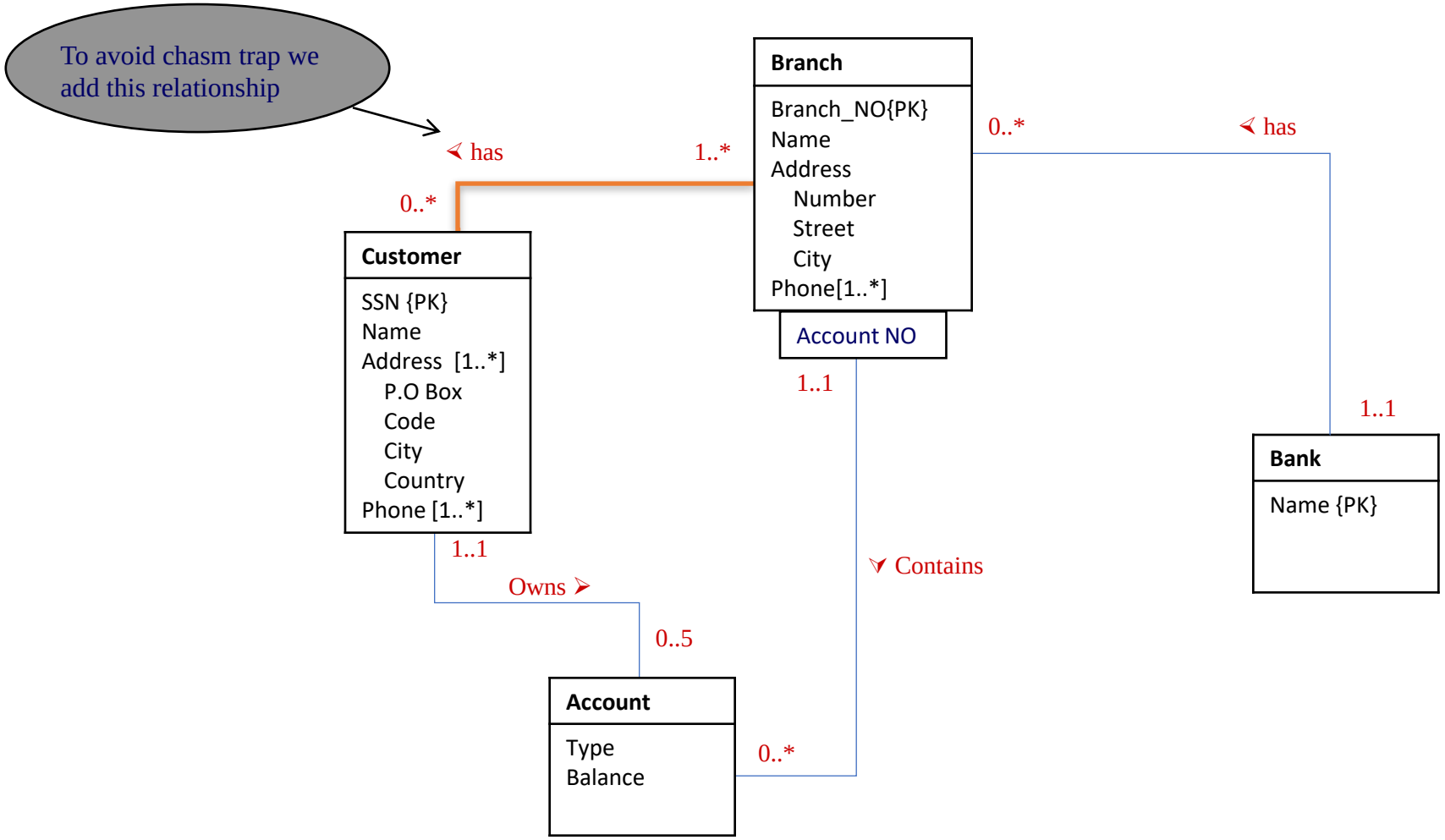
Banks Database :ER Case Study

- **Design an ER schema for this application, and draw an ER diagram for that schema.**
- **Specify key attributes of each entity type and structural constraints on each relationship type.**
- **Note any unspecified requirements, and make appropriate assumptions to make the specification complete.**

Banks Database :ER Case Study



Banks Database :ER Case Study



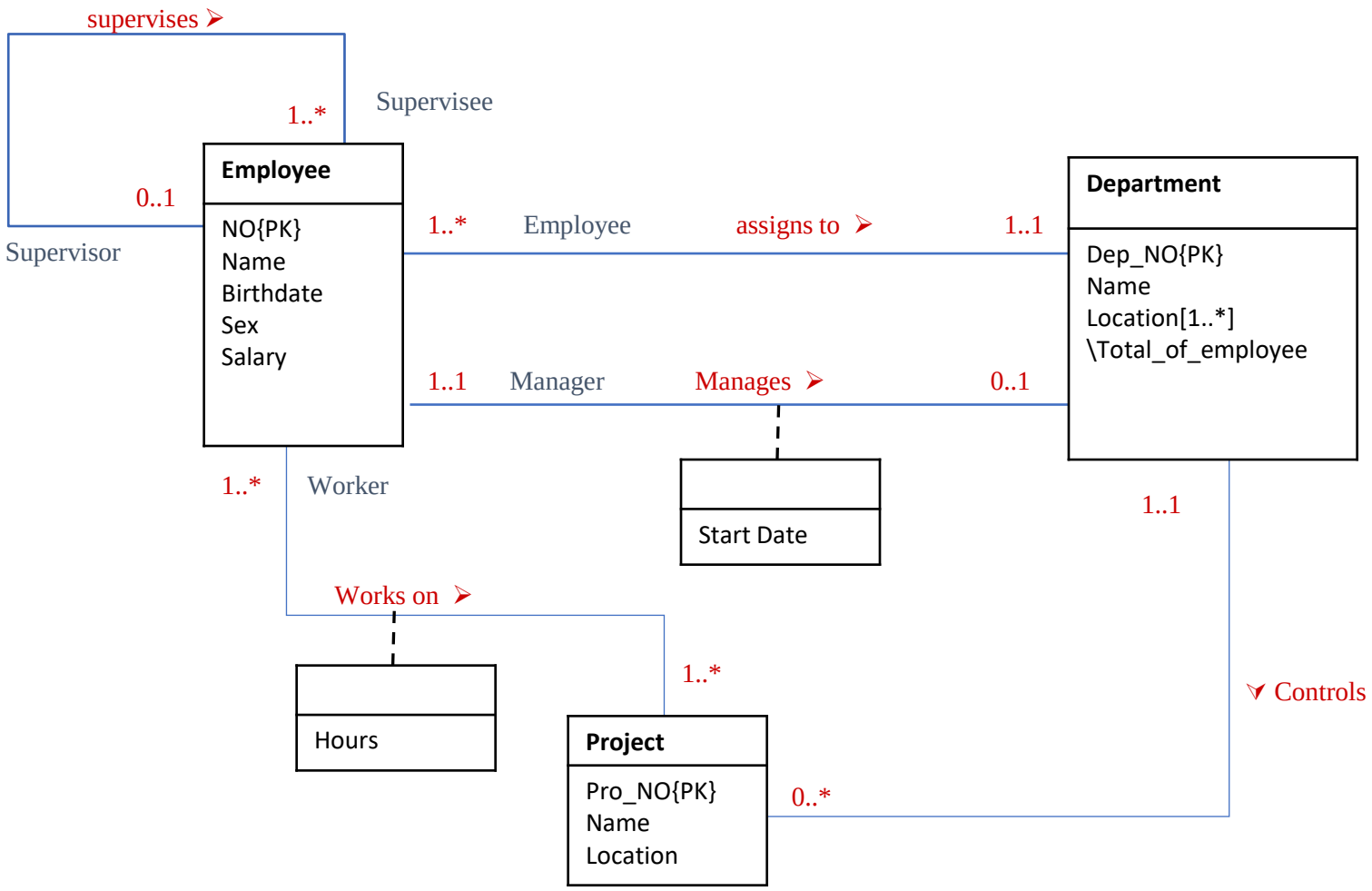
Organization :ER Case Study

- Organization made up of various departments, each having a name, identifying no., and an employee who is the manager. A department may be located in different places. Information about employee includes name, identification number, birth date, address, sex, and salary. Each employee is assigned to one department. The date the manager is appointed to a department is also tracked. Employees may be directly supervised by another employee. Each project within the organization is controlled by a department. Employees (not necessarily from the controlling dept.) are assigned to projects. Information about projects includes project name, no., and location. Hours spent by employees on each project are also kept.

Organization :ER Case Study

- draw an ER diagram to represent the data requirements as following:
  - Identify the main entity types.
  - Identify the main relationship types between the entity types.
  - Identify attributes and associate them with entity or relationship types.
  - Determine candidate and primary key attributes for each (strong) entity type.
  - Determine the multiplicity constraints for each relationship .State any assumptions necessary to support your design.

Organization :ER Case Study



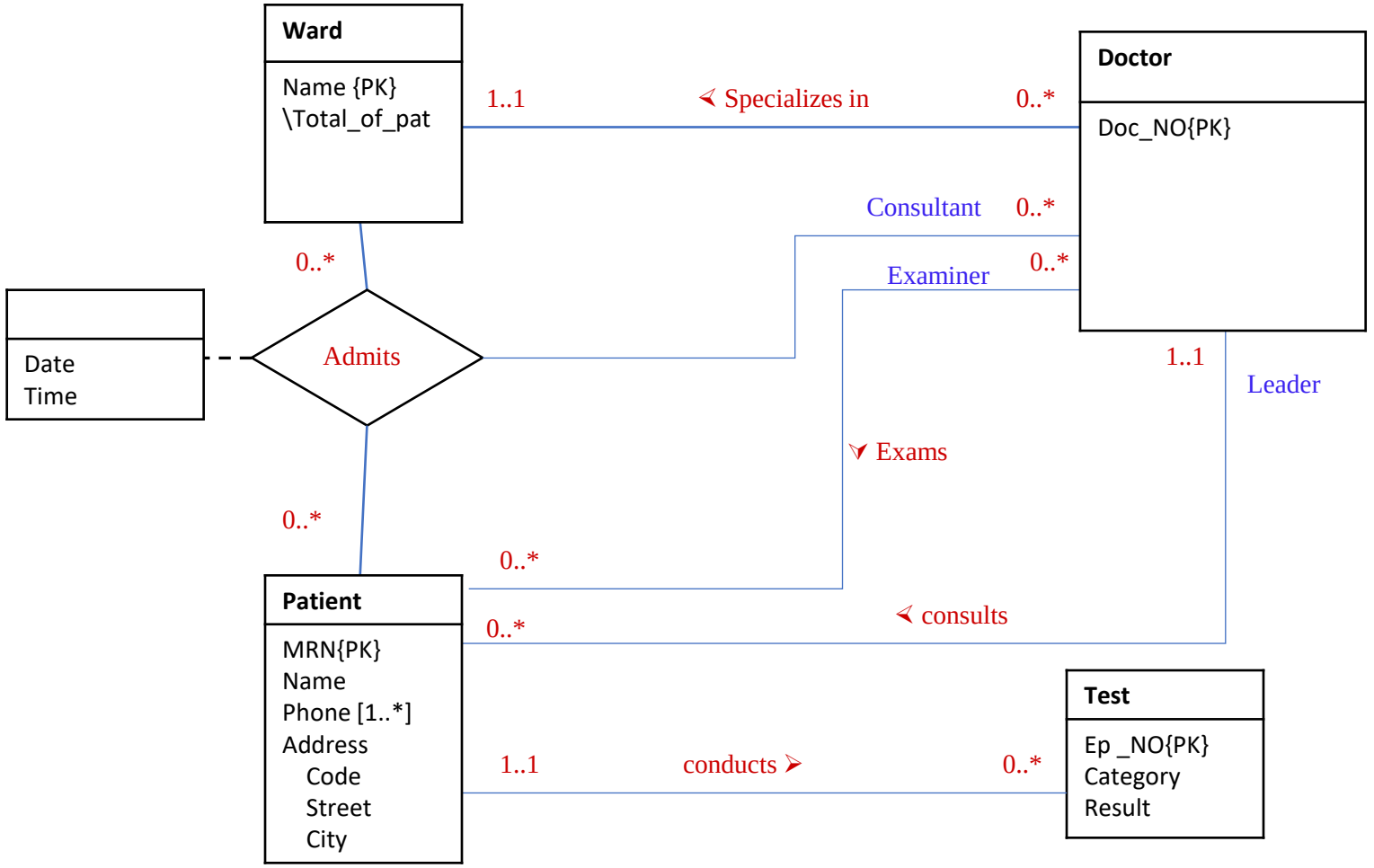
Hospital :ER Case Study

- A General Hospital consists of a number of specialized wards (such as Radiology, Oncology, etc) .Information about ward includes unique name, total numbers of current patients. Each ward hosts a number of patients, who were admitted by a consultant (doctors) employed by the Hospital. On admission, the date and time are kept. The personal details of every patient includes name, Medical Recode Number (MRN), set of phone and one address (city, street, code). A separate register is to be held to store the information of the tests undertaken. Each test has unique episode No. , category and the final result of test. Number of tests may be conducted for each patient. Doctors are specialists in a specific ward and may be leading consultants for a number of patients. Each patient is assigned to one leading consultant but may be examined by other doctors, if required.

Hospital :ER Case Study

- draw an ER diagram to represent the data requirements as following:
  - Identify the main entity types.
  - Identify the main relationship types between the entity types.
  - Identify attributes and associate them with entity or relationship types.
  - Determine candidate and primary key attributes for each (strong) entity type.
  - Determine the multiplicity constraints for each relationship .State any assumptions necessary to support your design.

Hospital :ER Case Study

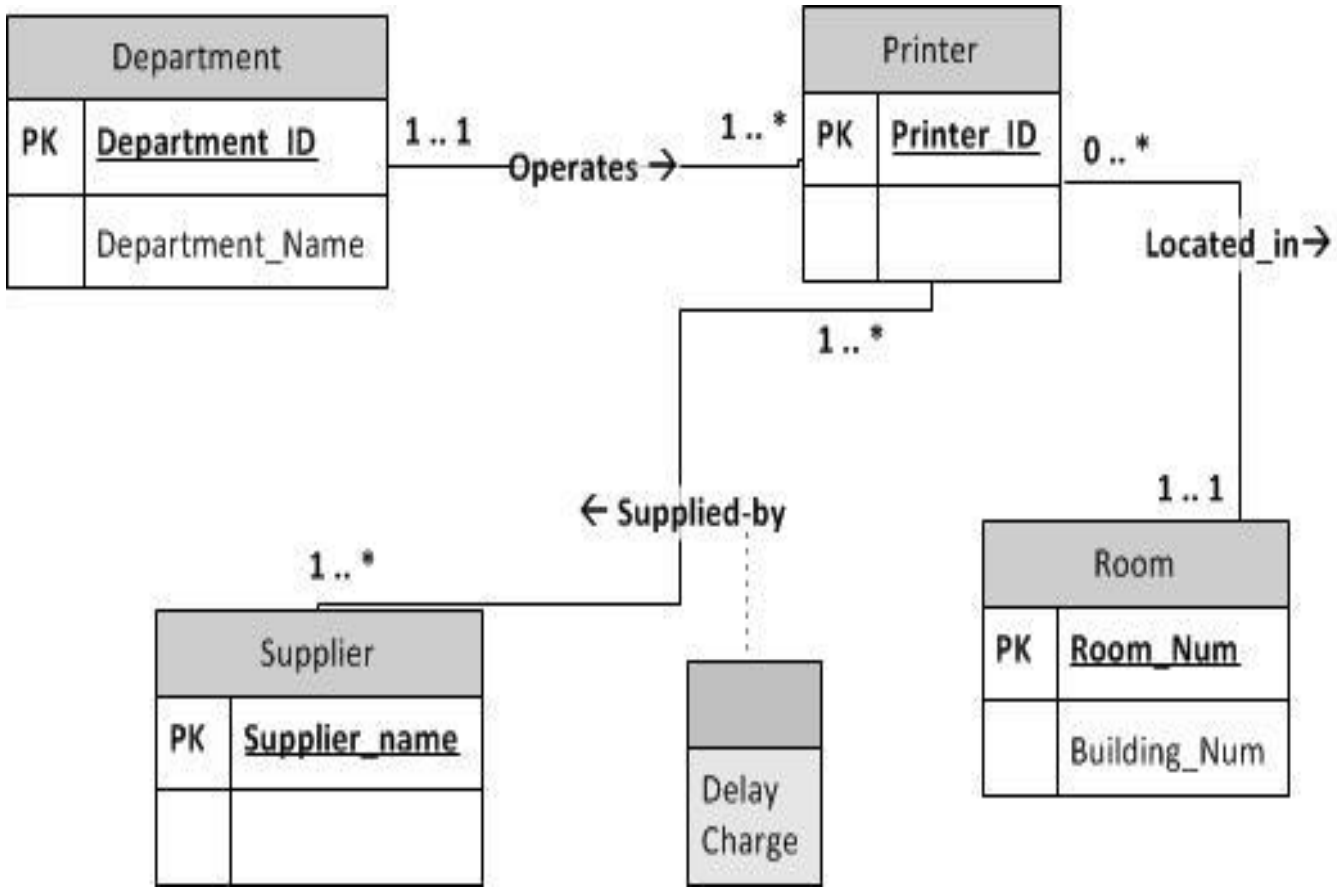


Department :ER Case Study

- Identify the entities and relationships for the following description and draw an ER diagram.

  Departments, identified by ID, operate a variety of printers, each located in a particular room in a particular building. Printers are supplied by a number of suppliers, identified by name, with each supplier charging a different price for a given printer, but also providing different delivery delays, measured in days. A given room can have any number of printers, including none.

Department :ER Case Study

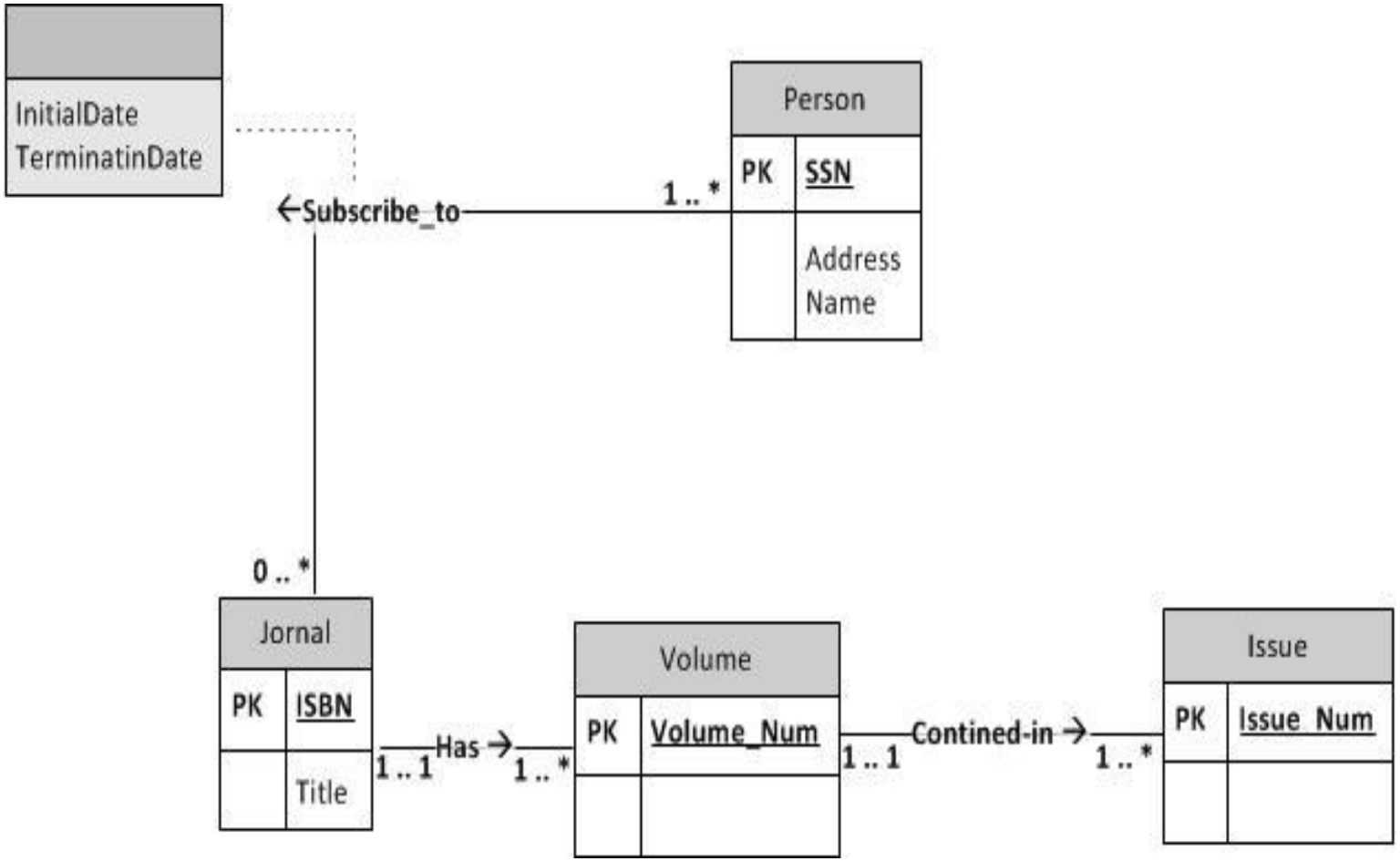


Journal :ER Case Study

- Identify the entities and relationships for the following description and draw an ER diagram.

  Persons, described by their name, SSN, and address, subscribe to various journals. Each journal, identified by a title and an ISBN, has a set of numbered volumes and each of these has a set of numbered issues. Subscribers have an initial subscription date and a termination date for each journal to which they subscribe.

Journal :ER Case Study



NORMALIZATION

Why we need normalization

- A large database defined as a single relation may result in data duplication. This repetition of data may result in:
  - Making relations very large.
  - It isn't easy to maintain and update data as it would involve searching many records in relation.
  - Wastage and poor utilization of disk space and resources.
  - The likelihood of errors and inconsistencies increases.
- So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties
- Normalization is a process of decomposing the relations into relations with fewer attributes.

What is normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

First Normal From

- A table is in the first normal form iff
  - The domain of each attribute contains only ***atomic values***, and
  - The value of each attribute contains only a ***single value*** from that domain.

In layman's terms. it means every column of your table should only contain *single values*

Example

- For a library

Patron ID	Borrowed books
C45	B33, B44, B55
C12	B56

1-NF Solution

Patron ID	Borrowed book
C45	B33
C45	B44
C45	B33
C12	B56

Example

- For an airline

Flight	Weekdays
UA59	Mo We Fr
UA73	Mo Tu We Th Fr

1NF Solution

Flight	Weekday
UA59	Mo
UA59	We
UA59	Fr
UA73	Mo
UA73	We
...	...

Implication for the ER model

- Watch for entities that can have multiple values for the same attribute
  - Phone numbers, …
- What about course schedules?
  - MW 5:30-7:00pm
    - Can treat them as *atomic time slots*

Functional dependency

Let X and Y be *sets* of attributes in a table *T*

- Y is **functionally dependent** on *X* in *T* **iff** for each set x $\in$ R.X there is precisely one corresponding set y$\in$ R.Y
- Y is **fully functional dependent** on *X* in *T* if *Y* is functional dependent on X and Y is not functional dependent on any proper subset of *X*

Example

- Book table

BookNo	Title	Author	Year
B1	Moby Dick	H. Melville	1851
B2	Lincoln	G. Vidal	1984

Author attribute is:

- ***functionally dependent*** on the pair { BookNo, Title}
- ***fully functionally dependent*** on BookNo

Why it matters

- table BorrowedBooks

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

Address attribute is

- ***functionally dependent*** on the pair { BookNo, Patron}
- ***fully functionally dependent*** on Patron

Problems

- Cannot insert new patrons in the system until they have borrowed books
  - ***Insertion anomaly***
- Must update all rows involving a given patron if he or she moves.
  - ***Update anomaly***
- Will lose information about patrons that have returned all the books they have borrowed
  - ***Deletion anomaly***

Armstrong inference rules (1974)

- ***Axioms:***
  - Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
  - Augmentation: if $X \rightarrow Y$, then $WX \rightarrow WY$
  - Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- ***Derived Rules:***
  - Union: if $X \rightarrow Y$ and $X \rightarrow Z$, the $X \rightarrow YZ$
  - Decomposition: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudotransitivity: if $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$

Armstrong inference rules (1974)

- Axioms are both
  - ***Sound:***
    when applied to a set of functional dependencies they only produce dependency tables that belong to the transitive closure of that set
  - ***Complete:***
    can produce all dependency tables that belong to the transitive closure of the set

Armstrong inference rules (1974)

- Three last rules can be derived from the first three (the axioms)
- Let us look at the ***union rule***:
  if X→Y and X→Z, the X→YZ
- Using the first three axioms, we have:
  - if X→Y, then XX→XY same as X→XY (2nd)
  - if X→Z, then YX→YZ same as XY→YZ (2nd)
  - if X→XY and XY→YZ, then X→YZ (3rd)

Second Normal Form

- A table is in 2NF iff
  - It is in 1NF and
  - no non-prime attribute is dependent on any proper subset of any candidate key of the table
- A ***non-prime attribute*** of a table is an attribute that is not a part of any candidate key of the table
- A ***candidate key*** is a minimal superkey

Example

- Library allows patrons to request books that are currently out

BookNo	Patron	PhoneNo
B3	J. Fisher	555-1234
B2	J. Fisher	555-1234
B2	M. Amer	555-4321

Example

- Candidate key is {BookNo, Patron}
- We have
  - Patron → PhoneNo
- Table is not 2NF
  - Potential for
    - Insertion anomalies
    - Update anomalies
    - Deletion anomalies

2NF Solution

- Put telephone number in separate Patron table

BookNo	Patron
B3	J. Fisher
B2	J. Fisher
B2	M. Amer

Patron	PhoneNo
J. Fisher	555-1234
M. Amer	555-4321

Third Normal Form

- A table is in 3NF iff
  - it is in 2NF and
  - all its attributes are determined only by its candidate keys and not by any non-prime attributes

Example

- Table BorrowedBooks

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

- Candidate key is BookNo
- Patron $\rightarrow$ Address

3NF Solution

- Put address in separate Patron table

BookNo	Patron	Due
B1	J. Fisher	3/2/15
B2	L. Perez	2/28/15

Patron	Address
J. Fisher	101 Main Street
L. Perez	202 Market Street

Another example

- Tournament winners

Tournament	Year	Winner	DOB
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 Sept. 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975

- Candidate key is {Tournament, Year}
- Winner →DOB

Boyce-Codd Normal Form

- Stricter form of 3NF
- A table *T* is in BCNF iff
  - for every one of its non-trivial dependencies $X \rightarrow Y$, *X* is a super key for *T*

- Most tables that are in 3NF also are in BCNF

Example

Manager	Project	Branch
Alice	Alpha	Austin
Alice	Delta	Austin
Carol	Alpha	Houston
Dean	Delta	Houston

- We can assume
  - Manager → Branch
  - {Project, Branch} → Manager

Example

Manager	Project	Branch
Alice	Alpha	Austin
Bob	Delta	Houston
Carol	Alpha	Houston
Alice	Delta	Austin

- Not in BCNF because Manager → Branch and Manager is not a superkey
- Will decomposition work?

A decomposition (I)

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston

- Two-table solution does not preserve the dependency {Project, Branch} → Manager

A decomposition (II)

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta
Dean	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston
Dean	Houston

- Cannot have two or more managers managing the same project at the same branch

Multivalued dependencies

- Assume the column headings in a table are divided into three disjoint groupings *X*, *Y*, and *Z*
- For a particular row, we can refer to the data beneath each group of headings as *x*, *y*, and *z* respectively

Multivalued dependencies

- A ***multivalued dependency*** X =>Y occurs if
  - For any x_c actually occurring in the table and the list of all the $x_c$yz combinations that occur
    in the table, we will find that x_c is associated with the same y entries regardless of z.
- A **t*rivial multivalued dependency*** *X* =>*Y* is one where either
  - *Y* is a subset of *X*, or
  - *Z* is empty (*X* ∪ *Y* has all column headings)

Fourth Normal Form

- A table is in 4NF iff
  - For every one of its non-trivial multivalued dependencies *X* => *Y*, *X* is either:
    - A candidate key or
    - A superset of a candidate key

Example from Wikipedia

Restaurant	Pizza	DeliveryArea
Pizza Milano	Thin crust	SW Houston
Pizza Milano	Thick crust	SW Houston
Pizza Firenze	Thin crust	NW Houston
Pizza Firenze	Thick crust	NW Houston
Pizza Milano	Thin crust	NW Houston
Pizza Milano	Thick crust	NW Houston

Discussion

- The table has no non-key attributes
  - Key is { Restaurant, Pizza, DeliveryArea}
- Two non-trivial multivalued dependencies
  - Restaurant => Pizza
  - Restaurant => DeliveryArea

  since each restaurant delivers the same pizzas to all its delivery areas

4NF Solution

- Two separate tables

Restaurant	DeliveryArea
Pizza Milano	SW Houston
Pizza Firenze	NW Houston
Pizza Milano	NW Houston

Restaurant	Pizza
Pizza Milano	Thin crust
Pizza Milano	Thick crust
Pizza Firenze	Thin crust
Pizza Firenze	Thick crust

Join dependency

- A table *T* is subject to a ***join dependency*** if it can always be recreated by ***joining*** multiple tables each having a subset of the attributes of T

- The join dependency is said to be ***trivial*** if one of the tables in the join has all the attributes of the table *T*
- *Notation: **{ A, B, …}* on *T*

Fifth normal form

- A table *T* is said to be 5NF iff
  - Every non-trivial join dependency in it is implied by its candidate keys

- A join dependency *{A, B, … Z} on T is implied by the candidate key(s) of T if and only if each of A, B, …, Z is a superkey for T

An example

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops

- Note that Circuit City sells Apple tablets and phones but only Toshiba laptops

A very bad decomposition

Store	*Product*
Circuit City	Tablets
Circuit City	Phones
Circuit City	Laptops
CompUSA	Laptops

Brand	*Product*
Apple	Tablets
Apple	Phones
Apple	Laptops
Toshiba	Laptops

- Let see what happens when we do a natural join

The result of the join

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Apple	Laptops
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops
CompUSA	Toshiba	Laptops

- Introduces two spurious tuples

A different table

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Apple	Laptops
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops

- Assume now that any store carrying a given brand and selling a product that is made by that brand will ***always*** carry that product

The same decomposition

Store	*Product*
Circuit City	Tablets
Circuit City	Phones
Circuit City	Laptops
CompUSA	Laptops

Brand	*Product*
Apple	Tablets
Apple	Phones
Apple	Laptops
Toshiba	Laptops

- Let see what happens when we do a natural join

The result of the join

Store	*Brand*	*Product*
Circuit City	Apple	Tablets
Circuit City	Apple	Phones
Circuit City	Apple	Laptops
Circuit City	Toshiba	Laptops
CompUSA	Apple	Laptops
CompUSA	Toshiba	Laptops

- Still one spurious tuple

The right decomposition

Store	*Product*
Circuit City	Tablets
Circuit City	Phones
Circuit City	Laptops
CompUSA	Laptops

Brand	*Product*
Apple	Tablets
Apple	Phones
Apple	Laptops
Toshiba	Laptops

Store	*Brand*
Circuit City	Apple
Circuit City	Toshiba
CompUSA	Apple

Conclusion

- The first "big" table was 5NF
- The second table was decomposable

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

An Example

Normalisation Example

- We have a table representing orders in an online store
- Each row represents an item on a particular order
- Primary key is {Order, Product}

- Columns
  - Order
  - Product
  - Quantity
  - UnitPrice
  - Customer
  - Address

Functional Dependencies

- Each order is for a single customer:
  - Order $\rightarrow$ Customer
- Each customer has a single address
  - Customer $\rightarrow$ Address
- Each product has a single price
  - Product $\rightarrow$ UnitPrice
- As Order $\rightarrow$ Customer and Customer $\rightarrow$ Address
  - Order $\rightarrow$ Address

2NF Solution (I)

- ***First decomposition***
  - First table

Order	Product	Quantity	UnitPrice

  - Second table

Order	Customer	Address

2NF Solution (II)

- ***Second decomposition***
  - First table

Order	Product	Quantity

  - Second table

Order	Customer	Address

  - Third table

Product	UnitPrice

3NF

- In second table

Order	Customer	Address

  - Customer $\rightarrow$ Address
- Split second table into

Order	Customer

Customer	Address

Normalisation to 2NF

- Second normal form means no partial dependencies on candidate keys
  - {Order} $\rightarrow$ {Customer, Address}
  - {Product} $\rightarrow$ {UnitPrice}

- To remove the first FD we project over

  {Order, Customer, Address} (R1)

and

{Order, Product, Quantity, UnitPrice} (R2)

Normalisation to 2NF

- R1 is now in 2NF, but there is still a partial FD in R2

  $\{Product\} \rightarrow \{UnitPrice\}$

- To remove this we project over

{Product, UnitPrice} (R3)

and

{Order, Product, Quantity} (R4)

Normalisation to 3NF

- R has now been split into 3 relations - R1, R3, and R4
  - R3 and R4 are in 3NF
  - R1 has a transitive FD on its key

- To remove

  {Order} $\rightarrow$ {Customer} $\rightarrow$ {Address}

- we project R1 over
  - {Order, Customer}
  - {Customer, Address}

Normalisation

- 1NF:
  - {Order, Product, Customer, Address, Quantity, UnitPrice}
- 2NF:
  - {Order, Customer, Address}, {Product, UnitPrice}, and {Order, Product, Quantity}
- 3NF:
  - {Product, UnitPrice}, {Order, Product, Quantity}, {Order, Customer}, and {Customer, Address}

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Top-Down Database design vs Bottom-Up Database design

- Top-Down Design: In top-down design, the database design starts with a high-level conceptual model and progresses to lower levels of detail, such as logical and physical data models. This approach is best suited for complex database systems that require a comprehensive understanding of the data requirements before the database is built.
- Bottom-Up Design: In bottom-up design, the database design starts with existing database structures and data and then progresses to higher levels of abstraction, such as logical and conceptual data models. This approach is best suited for organizations that have existing databases and need to integrate or improve them, rather than starting from scratch.

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)